AURORA-LM: AUTOENCODING UNIFIED REPRESENTATION FOR CONTINUOUS-LATENT DIFFUSION LANGUAGE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Continuous latent-space modeling has proved effective for image, video, and audio generation, while text generation is still dominated by models that directly predict discrete tokens. Existing continuous language models often inherit embedding spaces not designed for both generation and decoding, or compress autoencoded latents to ease diffusion, potentially sacrificing token-level fidelity. Instead of simplifying the representation to accommodate the generative model, we introduce **AURORA-LM**, a continuous-latent diffusion language model that preserves a high-capacity, decodable text latent and learns its distribution separately. A Query-based Encoder-Decoder organizes text into a prefix-aligned latent sequence. We then freeze the autoencoder and train a Block-causal Diffusion Transformer to model these full-width latents through blockwise flow matching. However, retaining these high-dimensional latents makes diffusion modeling more challenging. We address this difficulty by restricting only the noisy-input pathway through a low-rank bottleneck while retaining the full clean-latent prediction target, and calibrating the noise-level distribution to the latent width. Finally, we introduce self-trajectory consistency to bridge the gap between training on independently sampled noisy states and inference through iterative denoising. AURORA-LM achieves the best Gen-PPL and MAUVE on OpenWebText free generation and the best ROUGE scores on XSum conditional summarization among the evaluated continuous and diffusion-based language models. Scaling its denoiser to 1B parameters, it also outperforms a larger publicly released latent-diffusion language model under matched prompting and scoring.

1 INTRODUCTION

Continuous latent spaces have become a general interface for representation learning and generation (Rombach et al., 2022). Diffusion models have achieved remarkable quality in image, video, and audio synthesis (Ho et al., 2020; Song et al., 2021; Rombach et al., 2022; Ho et al., 2022; Kong et al., 2021), whereas modern language models rely on discrete token representations (Brown et al., 2020; Touvron et al., 2023; Yang et al., 2025). In current multimodal systems, continuous observations are commonly converted into discrete token sequences before they are consumed by a token-based Transformer backbone (Chameleon Team, 2025; Wang et al., 2024). The converse direction is therefore equally important: **instead of forcing every modality into discrete tokens, can language itself be represented and generated in a continuous latent space?**

Early attempts at continuous language generation largely begin from token embeddings (Li et al., 2022; Gulrajani & Hashimoto, 2023) or contextual features extracted by a pre-trained language encoder (Shabalin et al., 2025). However, the latent width and sequence organization of these representations are not explicitly designed to support both accurate text reconstruction and effective continuous generation.

To gain direct control over the representation itself, language autoencoders offer a more flexible alternative. These methods first map text into a continuous latent space and then learn a diffusion- or flow-based model over that space (Lovelace et al., 2023; Meshchaninov et al., 2025; Guo et al., 2026). This formulation allows the latent sequence to be compressed, smoothed, or organized hier-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

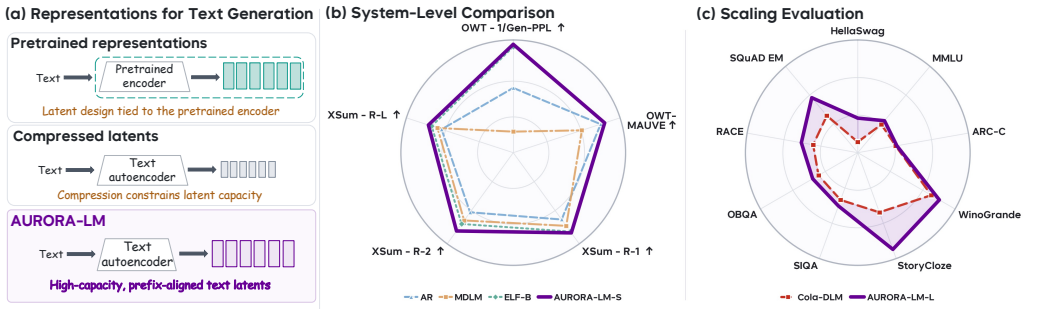

Figure 1: **Overview and evaluation of AURORA-LM.** (a) High-capacity, prefix-aligned text representations. (b) Generation results on OpenWebText and XSum; radial values are normalized per metric, with reciprocal Gen-PPL. (c) Comparison with a larger latent-diffusion language model across nine benchmarks.

archically, making its distribution more tractable for generation. However, compression can remove distinctions needed to recover the exact words, syntax, and local ordering. Conversely, retaining more information improves reconstruction but presents the latent prior with a wider and more complex distribution.

This tension suggests that a continuous language representation should not be designed solely to make its distribution easy to model. It must first preserve linguistic semantics and sufficient information for accurate token recovery. The latent generative model should accommodate this representation, without compressing the decoder-facing latent merely to make generation easier. This presents two connected challenges: constructing a faithfully decodable text representation and learning its latent distribution without sacrificing decoding fidelity.

To address these two connected challenges, we introduce **AURORA-LM**, a unified framework that decouples representation construction and distribution modeling into two stages. For the first challenge, we construct a high-capacity continuous latent sequence (Fig. 1(a)) from discrete text using a Query-based Encoder-Decoder. Its causal encoder reads progressively longer token prefixes, inducing a prefix order that supports blockwise left-to-right generation. A corresponding decoder maps latent prefixes back to tokens and is trained solely through text reconstruction.

Having established a decodable and causally ordered latent space, we turn to the second challenge: learning its induced distribution without sacrificing the representational capacity. Specifically, we freeze the autoencoder and train a block-causal denoiser on the resulting full-width latent sequences using flow matching. To model this distribution, we combine three designs. First, inspired by the direct clean-data prediction principle (Li & He, 2026), we apply a low-rank projection only to the noisy-latent input pathway while continuing to predict the full-width clean latent. Second, we calibrate the training-time noise allocation to latent width. Third, we introduce self-trajectory consistency, which aligns clean-latent predictions at neighboring states along the denoising trajectory and enables high-quality generation with fewer sampling steps.

As summarized in Fig. 1(b,c), AURORA-LM achieves the strongest performance among the evaluated methods on both OpenWebText (Gokaslan & Cohen, 2019) free generation and XSum (Narayan et al., 2018) conditional summarization. We further scale the AURORA-LM denoiser to 1B parameters. The resulting model outperforms a larger publicly released latent-diffusion language model across nine benchmarks under a shared prompting and scoring protocol.

In summary, our contributions are as follows:

- We introduce **AURORA-LM**, which constructs a high-capacity, prefix-aligned autoencoded text latent for continuous generation and token recovery, and learns its distribution separately with a block-causal diffusion transformer.
- Our controlled experiments show that wider text latents retain token-level information more robustly under corruption and, with calibrated noise allocation, achieve better generation quality.

- We introduce **self-trajectory consistency** to align clean-latent predictions before and after a model-induced denoising step, improving generation quality with the largest gains in the few-step regime.
- **AURORA-LM** achieves the best Gen-PPL and MAUVE on OpenWebText and ROUGE scores on XSum among the evaluated models. We further scale the denoiser to 1B parameters, surpassing a larger publicly released latent-diffusion language model across nine benchmarks under a shared evaluation protocol.

2 Related Work

2.1 Diffusion Models

Diffusion models (Ho et al., 2020; Song et al., 2021; Karras et al., 2022) learn to invert a forward process that progressively injects Gaussian noise. Flow matching (Lipman et al., 2023; Liu et al., 2022; Albergo & Vanden-Eijnden, 2023) recasts generation as learning a velocity field from noise to data. Consistency models (Song et al., 2023; Luo et al., 2023) map states along the same probability-flow ODE trajectory to a consistent clean endpoint. Consistency-FM (Yang et al., 2024) constrains endpoint and velocity predictions, while FlowConsist (Zhang et al., 2026) uses self-predicted marginal velocities for trajectory consistency. Our self-trajectory consistency regularizes text-latent flow matching by aligning clean-latent predictions across one model-induced Euler update.

2.2 Discrete Language Models

Autoregressive Language Models. Autoregressive language models factorize the joint distribution over a token sequence, generating each token conditioned on all preceding tokens. Next-token prediction provides a scalable self-supervised objective for large-scale language modeling (Brown et al., 2020; Hoffmann et al., 2022; OpenAI, 2023; Touvron et al., 2023; Grattafiori et al., 2024; DeepSeek-AI et al., 2025; Yang et al., 2025). However, standard autoregressive decoding generates one token at a time, limiting parallelism across output positions.

Discrete Diffusion Language Models. Discrete diffusion language models define a corruption–denoising process directly over token sequences (Austin et al., 2021). MDLM (Sahoo et al., 2024) expresses its objective as a mixture of masked language modeling losses, while SEDD (Lou et al., 2024) extends score matching to discrete spaces through a score entropy objective. BD3-LMs (Arriola et al., 2025) combine autoregressive generation across blocks with masked diffusion within each block. Duo (Sahoo et al., 2025) connects uniform-state discrete diffusion to Gaussian diffusion, enabling few-step generation through consistency distillation. CDLM (Amin et al., 2026) trains discrete denoisers with consistency constraints across exact posterior bridges between noise levels. LLaDA and Dream scale discrete diffusion to large language models (Nie et al., 2025; Ye et al., 2025), with further extensions to reasoning via reinforcement learning (Zhao et al., 2025) and code generation (Song et al., 2025).

AURORA-LM instead generates over a continuous latent space, with a learned encoder-decoder mapping between tokens and continuous representations.

2.3 Continuous Diffusion Language Models

Early continuous approaches perform denoising in token embedding spaces. Diffusion-LM (Li et al., 2022) demonstrated controllable generation by denoising word embeddings; DiffuSeq (Gong et al., 2023) extended this formulation to conditional sequence generation. PLAID (Gulrajani & Hashimoto, 2023) demonstrated competitive likelihood on standard benchmarks, while RE-PLAID (Yang et al., 2026) showed that continuous diffusion rivals discrete DLMs at scale. LangFlow (Chen et al., 2026) connected embedding-space diffusion to flow matching via Bregman divergence. CoDAR (Shen et al., 2026) addressed the token rounding bottleneck with a context-aware autoregressive decoder.

Another line of work models contextual representations or learned text latents. LD4LG (Lovelace et al., 2023) compresses pretrained encoder features with a Perceiver Resampler, while COSMOS (Meshchaninov et al., 2025) learns compressed, smooth latents through reconstruction and

noise augmentation. TEncDM (Shabalin et al., 2025) instead diffuses full-length pretrained contextual representations. ELF (Hu et al., 2026) performs flow matching over bidirectional contextual embeddings from a frozen T5 encoder and shares its denoising network with final token decoding. TextLDM (Jiang et al., 2026) aligns a Transformer VAE with frozen Qwen3 representations, then freezes its encoder for flow-matching training. Cola-DLM (Guo et al., 2026) pretrains a KL-regularized VAE before jointly optimizing the autoencoder and flow-matching model. LDLM (Meshchaninov et al., 2026) jointly learns a latent encoder, diffusion model, and decoder on top of frozen language-model features.

AURORA-LM constructs a high-capacity, prefix-aligned text latent through reconstruction, without relying on pretrained contextual features. We preserve decoder-facing capacity for accurate token recovery rather than reduce it solely for prior tractability. The autoencoder is then frozen, allowing diffusion-side design choices to be studied independently of representation learning.

3 The AURORA-LM Framework

AURORA-LM is organized around two learning problems: constructing a continuous text representation that can be decoded back into tokens, and modeling its generative distribution. The Query-based Encoder-Decoder constructs a high-capacity, causally ordered latent representation; we then freeze the autoencoder and train a block-causal denoiser on its standardized outputs through block-wise flow matching (Fig. 2).

3.1 Preliminaries

Diffusion models generate data by learning to reverse a gradual transformation from clean samples to noise (Ho et al., 2020; Song et al., 2021). We adopt the flow-matching formulation (Lipman et al., 2023; Liu et al., 2022) with straight interpolating paths. Given a clean sample x_0 and Gaussian noise $\varepsilon \sim \mathcal{N}(0, I)$, we define the linear probability path as:

$$x_t = (1-t)x_0 + t\varepsilon, \qquad t \in [0,1]. \tag{1}$$

The path interpolates from data at $t = 0$ to noise at $t = 1$. A neural network can then learn the reverse dynamics by predicting the path velocity or the clean endpoint x_0. AURORA-LM adopts clean-endpoint prediction in its final configuration, while Sec. 4.2 evaluates the alternative parameterizations. In AURORA-LM, the clean sample is the continuous sequence produced by the frozen encoder after standardization.

3.2 Continuous Text Latent Construction

Our goal is to construct a continuous text representation that serves as an interface between continuous generative modeling and discrete token decoding. Such a representation must preserve sufficient information for accurate token decoding and organize its latent positions in a causal order.

Latent Sequence. Given a token sequence $w_{1:L}$ of length L, we instantiate a query-based autoencoder that constructs $z_{\text{enc}} \in \mathbb{R}^{N\times D}$ with channel width D. The encoder output projection sets D, while the decoder input projection maps these latents to the decoder hidden width; both are omitted from the attention notation below. Latent queries allow the output sequence length N to vary independently of the input length L. Our main models retain the full sequence length, $N = L$; further experiments on reduced-length variants are reported in Appendix A.3.6, with their general formulation in Appendix A.1.1.

Prefix-aligned Encoding and Decoding. The encoder initializes N latent queries from a shared learnable vector. The decoder uses L queries initialized from a second shared learnable vector and maps latent prefixes to token logits. Let $z^{(\ell)}_{\text{enc},i}$ and $h^{(\ell)}_j$ denote the encoder and decoder query states at layer ℓ, and let E denote the trainable token embedding matrix. For $N = L$, we enforce prefix alignment through the following residual attention updates:

$$\begin{aligned} \widetilde{z}^{(\ell)}_{\text{enc},i} &= z^{(\ell)}_{\text{enc},i} + \mathrm{MHA}^{(\ell)}_{\text{enc}}\Big(z^{(\ell)}_{\text{enc},i},\, [z^{(\ell)}_{\text{enc},1:i}; E[w_{1:i}]]\Big), \\ \widetilde{h}^{(\ell)}_j &= h^{(\ell)}_j + \mathrm{MHA}^{(\ell)}_{\text{dec}}\Big(h^{(\ell)}_j,\, [h^{(\ell)}_{1:j}; z_{\text{enc},1:j}]\Big). \end{aligned} \tag{2}$$

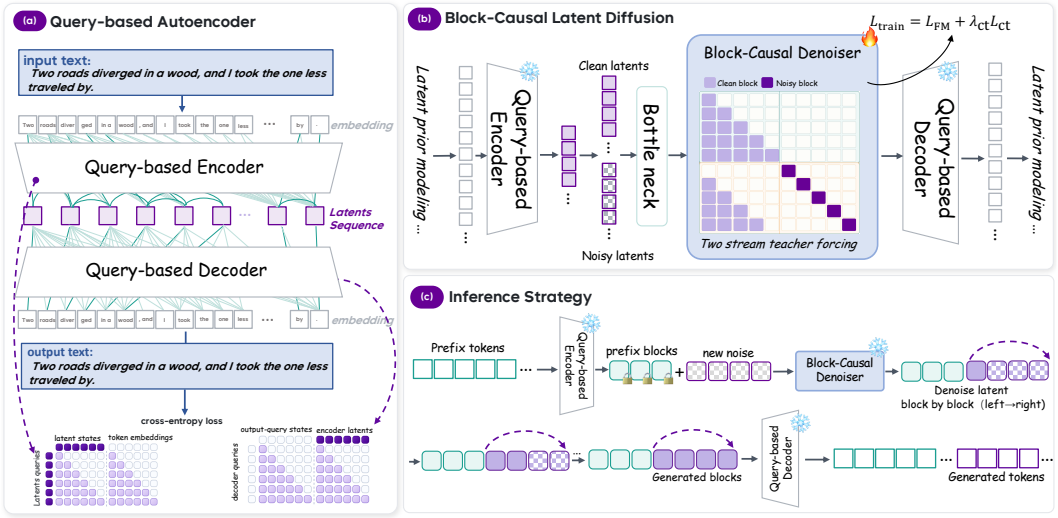

Figure 2: **Overview pipeline of AURORA-LM.** (a) The Query-based Encoder-Decoder constructs an ordered text latent sequence from causally expanding input prefixes and maps it back to token logits via a query decoder. (b) A block-causal denoiser parameterizes the latent prior and is trained with flow matching. (c) At inference, latent blocks are generated from left to right with KV caching of the completed prefix.

Here, the second argument of MHA provides the keys and values, and $[\,;\,]$ denotes sequence concatenation. Encoder position i depends only on the token prefix $w_{1:i}$. The decoder at position j is restricted to $z_{\mathrm{enc},1:j}$ and cannot access future latents. Because it does not receive the original tokens as input, it must recover w_j from this latent prefix. This matched visibility allows each generated latent block to be decoded using only the corresponding latent prefix. Full layer updates are given in Appendix A.1.1.

Autoencoder Training. We train the encoder and decoder jointly with token-level cross-entropy and dropout regularization (Meshchaninov et al., 2025) (Appendix A.1.2). After training, we disable dropout and freeze the autoencoder, fixing the decoder-facing representation for the diffusion model to learn its distribution.

3.3 Block-Causal Modeling of Continuous Text Latents

The remaining task is to learn the distribution of these representations so that the model can generate new latent sequences without access to the underlying text. We standardize the frozen encoder outputs as $z = (z_{\mathrm{enc}} - \mu) \oslash s$, where $\mu, s \in \mathbb{R}^D$ are the fixed per-channel mean and standard deviation and $\oslash$ denotes element-wise division. The latent prior $p_\theta(z)$ models the resulting empirical distribution $q_{\mathcal{E}}(z)$.

Block-wise Factorization. Modeling $p_\theta(z)$ as a single joint denoising problem would generate every latent position at once, leaving the causal organization learned by the autoencoder unused. At the other extreme, generating one latent position at a time would preserve the order but discard most of the parallelism offered by continuous denoising. We therefore adopt an intermediate block-causal formulation that performs left-to-right causal generation across latent blocks while jointly denoising all positions within each block. We call this intermediate organization a *block-causal factorization*.

For block size Q, we partition z into $B = \lceil N/Q \rceil$ contiguous blocks $\alpha = (\alpha^{(1)}, \ldots, \alpha^{(B)})$, and the prior is factorized as

$$p_\theta(\alpha) = \prod_{b=1}^{B} p_\theta\left(\alpha^{(b)} \mid \alpha^{(<b)}\right), \qquad \alpha^{(<b)} = (\alpha^{(1)}, \ldots, \alpha^{(b-1)}). \tag{3}$$

Because the encoder has organized latent prefixes to represent text prefixes, $\alpha^{(<b)}$ represents the textual context preceding block b.

Blockwise Flow Matching. We instantiate Eq. (1) for each block as $\alpha_t^{(b)} = (1-t)\alpha^{(b)} + t\varepsilon^{(b)}$, with $\varepsilon^{(b)} \sim \mathcal{N}(0, I)$. The block-causal denoiser f_θ predicts the clean endpoint as $\hat{\alpha}_\theta^{(b)} = f_\theta(\alpha_t^{(b)}, t; \alpha^{(<b)})$, using clean encoder blocks as prefix context during training. At inference, preceding blocks are generated sequentially; during training, clean encoder blocks are available as prefix context for all targets at once. Target blocks are independently corrupted; each can attend only to its own noisy positions and the preceding clean blocks. Clean blocks attend only to themselves and preceding clean blocks. This two-stream mask allows all block conditionals to be evaluated in parallel.

Concatenating the predicted blocks gives $\hat{z}_\theta$. With valid target latent positions $\mathcal{J}$ and noise levels sampled from $\pi(t)$, we minimize

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{\substack{z\sim q_\mathcal{E},\, t\sim\pi,\\ \varepsilon\sim\mathcal{N}(0,I)}} \left[\frac{1}{D|\mathcal{J}|} \sum_{i\in\mathcal{J}} \|\hat{z}_{\theta,i} - z_i\|_2^2 \right]. \tag{4}$$

Self-conditioning. We use self-conditioning (Chen et al., 2023), which feeds the clean-latent prediction from the previous sampling step back into the model as an additional input. During training, an auxiliary forward pass provides a detached draft prediction with probability p_{sc} (Table 4); otherwise, the self-conditioning input is omitted. We omit this additional input from the prediction notation above.

3.4 Learning the Full-Width Latent Distribution

The latent representation retains its channel capacity for token decoding, so the diffusion model must recover a wide, high-dimensional target from noisy inputs.

Noisy-latent Input Bottleneck. The frozen decoder expects a clean latent of width D, but the denoiser can process its noisy input through a narrower pathway. At large noise levels, much of the variation in $\alpha_t^{(b)}$ reflects Gaussian corruption rather than structure useful for recovering the clean block. Following JiT (Li & He, 2026), we use two linear layers for the noisy-input projection: the first reduces the width from D to D_b, and the second maps it to the Transformer hidden width H. This restricts only the noisy-input pathway; the output head still predicts the full D-dimensional clean block required by the frozen decoder. We study the choice of D_b in Sec. 4.2.

Noise-level Allocation. The noise-level distribution $\pi(t)$ determines how training samples are allocated along the diffusion path. We use a tan-d distribution, whose parameter d controls the allocation across noise levels, and select d for each latent width (Appendix A.1.4).

3.5 Self-trajectory Consistency

The flow-matching objective trains the diffusion model on independently sampled noisy states, whereas inference repeatedly evaluates it along a single denoising trajectory.

Self-conditioning provides the previous clean-latent estimate as an input, but does not explicitly enforce consistency across successive denoising states. We introduce *self-trajectory consistency* to align clean-latent predictions before and after a model-induced Euler update.

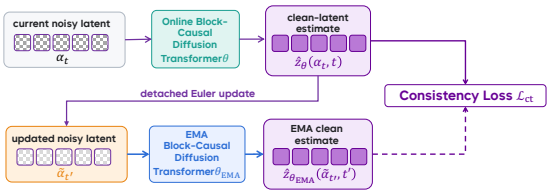

Figure 3: **Overview of self-trajectory consistency.** Self-trajectory consistency aligns neighboring clean-latent predictions along a model-induced update.

With clean prefix context fixed, we obtain $t' = \mathrm{Prev}_S(t)$ by one step toward lower noise in the schedule coordinate, using a training budget S (Appendix A.1.6). Let α_t denote the concatenated noisy target blocks and $\hat{z}_\theta(\alpha_t, t)$ their predicted clean latents, with conditioning inputs omitted. One Euler step gives

$$\widetilde{\alpha}_{t'} = \frac{t'}{t}\alpha_t + \left(1 - \frac{t'}{t}\right) \mathrm{sg}(\hat{z}_\theta(\alpha_t, t)), \tag{5}$$

where $\mathrm{sg}(\cdot)$ stops gradients through the update. An exponential moving average (EMA) model θ_{EMA} predicts a detached target at the updated state, using the detached current prediction for self-conditioning when enabled:

$$\mathcal{L}_{\mathrm{ct}} = \mathbb{E}_{z\sim q_{\mathcal{E}},\, t\sim\pi,\, \varepsilon\sim\mathcal{N}(0,I)} \left[\frac{1}{D|\mathcal{J}|} \sum_{i\in\mathcal{J}} \|\hat{z}_\theta(\alpha_t, t)_i - \mathrm{sg}(\hat{z}_{\theta_{\mathrm{EMA}}}(\widetilde{\alpha}_{t'}, t')_i)\|_2^2 \right]. \tag{6}$$

The complete prior objective is $\mathcal{L}_{\mathrm{train}} = \mathcal{L}_{\mathrm{FM}} + \lambda_{\mathrm{ct}}\mathcal{L}_{\mathrm{ct}}$. The flow-matching objective anchors each prediction to the clean training target, while the consistency objective aligns predictions at neighboring states along the sampling trajectory. The schedule configuration is given in Appendix A.1.6.

3.6 Latent Generation and Text Decoding

We generate latent blocks from left to right, starting from an empty prefix or a prompt prefix obtained with the frozen encoder and training-time standardization. Each block is denoised from Gaussian noise at $t = 1$ to $t = 0$ using ODE or SDE updates conditioned on the prefix, then appended to it. We inverse-standardize the generated latents as $z_{\mathrm{dec}} = \mu + s \odot \hat{z}$, where $\odot$ denotes element-wise multiplication, and decode them into token logits with the frozen query decoder.

Guidance. For unconditional generation, we use the previous solver step's clean-latent estimate as the self-conditioning signal. When available, self-conditioning classifier-free guidance (SC-CFG) combines predictions with and without this signal. For prompt-conditioned generation, standard classifier-free guidance (Ho & Salimans, 2022) combines predictions with and without the prompt prefix. Both operate on clean-latent predictions before the solver update (Appendix A.1.7).

4 Experiments

We evaluate AURORA-LM-S (130M) on OpenWebText free generation and XSum summarization, and AURORA-LM-L (1.01B) on nine language benchmarks. The latent width of AURORA-LM-S is $D = 768$. Parameter counts for our models refer to the denoiser and exclude the autoencoder. All experiments use Ascend NPUs; training and evaluation details are in Appendix A.2.

4.1 Generation Performance

Unconditional generation. We train AURORA-LM-S on 1,024-token OpenWebText sequences (Gokaslan & Cohen, 2019), following Duo's (Sahoo et al., 2025) data split and preprocessing (Appendix A.2.1). We evaluate released autoregressive, discrete-diffusion, and continuous-generation baselines without retraining. We measure generation perplexity (Gen-PPL) with GPT-2 Large (Radford et al., 2019), lexical diversity with token-unigram entropy, and distributional similarity with MAUVE (Pillutla et al., 2021). AURORA-LM-S uses 64-step SDE-DPM++ (Lu et al., 2025) sampling with SC-CFG scale 4; baseline settings are in Appendix A.3.7. AURORA-LM-S achieves the lowest Gen-PPL among the evaluated models and the highest MAUVE under the shared evaluation protocol (Table 1(a)).

Conditional summarization. To test conditional generation, we evaluate AURORA-LM-S on XSum (Narayan et al., 2018), conditioning on the source article and using 32-step ODE sampling with CFG scale 2. AURORA-LM-S achieves the highest ROUGE-1/2/L (Lin, 2004) in Table 1(b).

Larger-scale evaluation. We scale the denoiser to 1.01B parameters and train AURORA-LM-L on a pretraining mixture constructed from publicly available sources with approximately 1,500 EFLOPs of estimated total compute. Data and training details are provided in Appendix A.2. We compare with released Cola-DLM (Guo et al., 2026) (1.8B denoiser) on the same examples. Both models generate answers from shared prompts and use the same answer-matching procedure. Sample selection and model-specific sampling settings are detailed in Appendices A.2.8 and A.2.5, respectively. AURORA-LM-L scores higher on all nine benchmarks, averaging 32.6 versus 25.1 (Table 2).

4.2 Ablations and Analysis

We study latent capacity and its modeling on OpenWebText sequences of up to 128 tokens (OWT128), without sequence compression. Unless otherwise specified, we use 32-step ODE sam-

Table 1: **Free generation and conditional summarization.** Except for cited results, scores in this table are averaged over five generation seeds, with 1,000 samples per seed for OpenWebText. XSum baseline scores marked with † are taken from Hu et al. (2026). Parameter counts follow model-specific conventions (Table 6): AURORA counts the denoiser only, whereas ELF includes its output head.

(a) OpenWebText

Model	Params.	Gen-PPL↓	Entropy↑	MAUVE↑
AR	85M	39.40	5.605	0.851
Duo	92M	86.57	5.566	0.704
Duo-distilled	92M	78.22	5.574	0.715
SEDD	92M	119.82	5.644	0.693
MDLM	92M	121.36	**5.664**	0.668
ELF-B	105M	24.11	5.155	—*
AURORA-LM-S	130M	**23.56**	5.241	**0.890**

(b) XSum

Model	R-1↑	R-2↑	R-L↑
ELF-B†	36.0	12.2	27.8
AR†	30.5	10.2	24.4
MDLM†	33.4	11.6	25.8
Duo†	31.4	10.1	25.0
E2D2†	28.4	8.3	22.0
SeqDiffuSeq†	19.3	1.7	14.1
AURORA-LM-S	**36.6**	**13.4**	**28.9**

*ELF-B's MAUVE uses different reference data and truncation; its result is reported separately in Appendix A.2.7.

Table 2: **Evaluation at the 1B scale.** Scores are accuracy on eight multiple-choice tasks and normalized exact match on SQuAD, expressed as percentages. Avg is the mean across nine tasks; model sizes refer to denoisers.

Model	Avg↑	MMLU	ARC-C	OBQA	Hella Swag	Wino Grande	Story Cloze	SIQA	RACE	SQuAD EM
Cola-DLM (1.8B)	25.1	19.6	20.6	24.2	5.7	45.3	33.8	26.8	24.2	25.7
AURORA-LM-L	**32.6**	**22.2**	**21.2**	**27.8**	**18.4**	**50.3**	**54.8**	**30.2**	**30.6**	**38.2**

pling without guidance and average MAUVE across five generation seeds, with 1,000 samples per seed.

Latent capacity. We first examine whether wider latents benefit both token recovery and generation. We train autoencoders with latent widths $D \in \{128, 256, 512, 1024\}$ using fixed Transformer backbones and varying the latent-width projections (Appendix A.1.1). We corrupt encoder outputs as $x_\sigma = (1-\sigma)z + \sigma\epsilon$, $\epsilon \sim \mathcal{N}(0, I)$, and decode without denoising. All widths achieve near-perfect clean reconstruction, but wider latents tolerate stronger corruption (Figure 4(a)). Clean reconstruction alone therefore provides an incomplete measure of representation capacity. We next examine whether wider latents also benefit generation, sweeping the tan-d shift (Hoogeboom et al., 2023) over $d \in \{1, 3, 5, 7\}$ for each width. The preferred noise allocation varies with latent width. After calibrating the schedule for each width, the best MAUVE increases from 0.594 at $D = 128$ to 0.839 at $D = 1024$ (Figure 4(b)). We fix $D = 1024$ in the following ablations to study the modeling of high-capacity latents.

Noisy-input bottleneck. With the clean target fixed at $D = 1024$, varying only D_b yields the highest mean MAUVE over 16/32/64 steps at $D_b = 128$ (Figure 4(c)). The model can therefore retain the full-width clean target without processing the noisy input at full width.

Noise allocation. To examine noise allocation for this target, we vary d for tan-d and μ for logit-normal sampling at $s \in \{0.5, 1.0\}$. Within each of the three schedule families, MAUVE increases as more training probability is assigned to $\sigma > 0.7$ (Figure 5(a)). This trend supports high-noise allocation for the selected wide latent target. We use $d = 5$ for component ablations and $d = 7$ in the final system (Appendix A.3.3).

Prediction target and loss space. With noise allocation fixed, we compare clean-latent (x_0) and velocity (v) prediction with losses in either space. Direct x_0 prediction with an x_0-space loss achieves the highest MAUVE at both input widths (Figure 5(b)). For x_0-prediction, switching to a v-space loss reweights the reconstruction error by $1/\sigma^2$, emphasizing low-noise states. Under the adopted tan-d schedule, this reweighting reduces MAUVE from 0.815 to 0.059 at $D_b = 128$ (Appendix A.1.5).

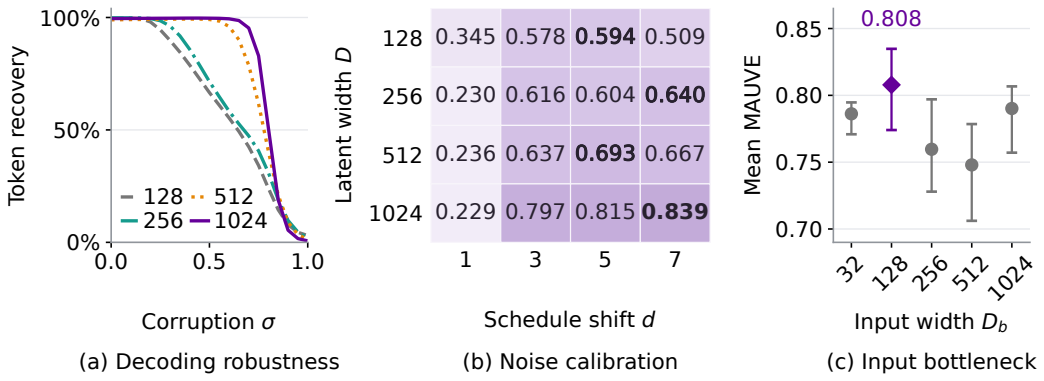

Figure 4: **Latent capacity and prior modeling on OWT128.** (a) Token recovery under corruption, without denoising. (b) MAUVE by latent width and schedule shift; row maxima are highlighted and the overall maximum is bold. (c) Noisy-input bottleneck ($D = 1024$): mean MAUVE over 16/32/64 steps; bars show the min–max range across budgets. Seed standard deviations are in Appendix A.3.

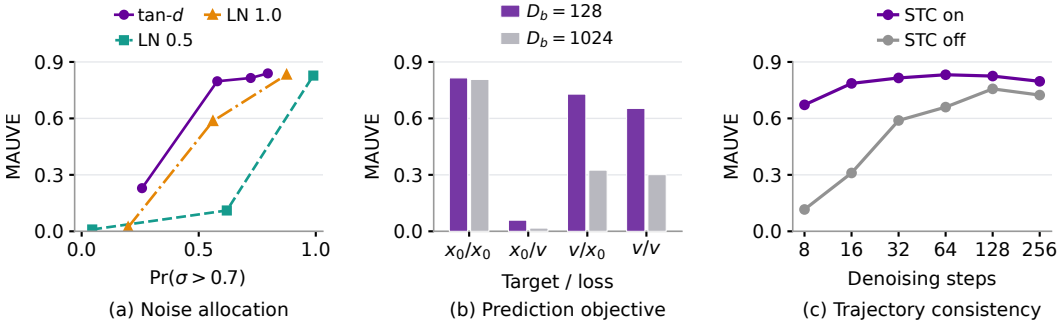

Figure 5: **Training choices and trajectory consistency on OWT128** ($D = 1024$). (a) Noise allocation; LN denotes logit-normal sampling. (b) Prediction target/loss combinations at 32 steps. (c) STC across sampling budgets. All panels report mean MAUVE with ODE sampling and no guidance; settings and seed standard deviations are in Appendix A.3.

Self-trajectory consistency. To evaluate the benefit of aligning clean-latent predictions along model-induced updates (Sec. 3.5), we compare models trained with and without STC across denoising-step budgets. STC improves MAUVE at every tested budget, from 0.116 to 0.672 at 8 steps, with larger gains in the few-step regime (Figure 5(c)).

Full ablation results and additional analyses of sequence compression and block size are provided in Appendix A.3.

5 Conclusion

We presented AURORA-LM, a continuous-latent diffusion language model that decouples representation learning from distribution modeling. A high-capacity, causally structured text latent supports both token recovery and blockwise generation. We retain the full-width clean target while adapting the noisy-input pathway and noise allocation, and use self-trajectory consistency to align predictions across denoising steps. Experiments demonstrate strong results on OpenWebText generation and XSum summarization; trained on a pretraining mixture constructed from publicly available sources, AURORA-LM with a 1B-parameter denoiser also outperforms a larger publicly released latent-diffusion language model across nine benchmarks. Together, these results show that a high-capacity, decodable text representation can serve as an effective interface between diffusion-based generation and discrete token decoding.

AI USE STATEMENT

We used generative AI tools to polish the writing and identify relevant prior work. We also sought suggestions on the layout and presentation of our figures and feedback on the evaluation protocols. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

AURORA-LM is trained on web and other English-language text sources described in Appendix A.2.1. Such data may contain biases, harmful content, or personal information, which generative models can reproduce. The evaluations in this paper measure reconstruction, generation quality, and benchmark performance; they do not assess privacy leakage or the safety of model outputs in deployment. These risks require separate assessment before broader use.

REPRODUCIBILITY STATEMENT

Section 3 defines the autoencoder, block-causal latent model, and training objectives; Appendix A.1 gives the corresponding implementation details. Appendix A.2 documents data preprocessing, model and training configurations, generation settings, baseline reproduction, and evaluation protocols. Appendix A.3 provides additional numerical studies and inference sweeps, while Appendices A.4–A.6 provide generated examples and benchmark demonstrations.

REFERENCES

Michael S. Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants, 2023.

Hasan Amin, Yuan Gao, Yaser Souri, Subhojit Som, Ming Yin, Rajiv Khanna, and Xia Song. Consistent Diffusion Language Models. arXiv preprint arXiv:2605.00161, 2026.

Marianne Arriola, Aaron Kerem Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block Diffusion: Interpolating Between Autoregressive and Diffusion Language Models. In *International Conference on Learning Representations*, volume 2025, pp. 50726–50753, 2025.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured Denoising Diffusion Models in Discrete State-Spaces. In *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993. Curran Associates, Inc., 2021.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Chameleon Team. Chameleon: Mixed-Modal Early-Fusion Foundation Models. arXiv preprint arXiv:2405.09818, 2025.

Ting Chen, Ruixiang Zhang, and Geoffrey Hinton. Analog bits: Generating discrete data using diffusion models with self-conditioning, 2023.

Yuxin Chen, Chumeng Liang, Hangke Sui, Ruihan Guo, Chaoran Cheng, Jiaxuan You, and Ge Liu. LangFlow: Continuous Diffusion Rivals Discrete in Language Modeling. arXiv preprint arXiv:2604.11748, 2026.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. arXiv preprint arXiv:1803.05457, 2018.

DeepSeek-AI, Aixin Liu, Bei Feng, et al. Deepseek-v3 technical report, 2025.

Aaron Gokaslan and Vanya Cohen. OpenWebText Corpus, 2019.

Shansan Gong, Mukai Li, Jiangtao Feng, Zhiyong Wu, and Lingpeng Kong. Diffuseq: Sequence to sequence text generation with diffusion models, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models, 2024.

Ishaan Gulrajani and Tatsunori B Hashimoto. Likelihood-Based Diffusion Language Models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 16693–16715. Curran Associates, Inc., 2023.

Hongcan Guo, Qinyu Zhao, Yian Zhao, Shen Nie, Rui Zhu, Qiushan Guo, Feng Wang, Tao Yang, Hengshuang Zhao, Guoqiang Wei, and Yan Zeng. Continuous Latent Diffusion Language Model. arXiv preprint arXiv:2605.06548, 2026.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-Key Normalization for Transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4246–4253, Online, 2020. Association for Computational Linguistics.

Jonathan Ho and Tim Salimans. Classifier-Free Diffusion Guidance. arXiv preprint arXiv:2207.12598, 2022.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851. Curran Associates, Inc., 2020.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video Diffusion Models. arXiv preprint arXiv:2204.03458, 2022.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training Compute-Optimal Large Language Models. arXiv preprint arXiv:2203.15556, 2022.

Emiel Hoogeboom, Jonathan Heek, and Tim Salimans. simple diffusion: End-to-end diffusion for high resolution images. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 13213–13232. PMLR, 2023.

Keya Hu, Linlu Qiu, Yiyang Lu, Hanhong Zhao, Tianhong Li, Yoon Kim, Jacob Andreas, and Kaiming He. ELF: Embedded Language Flows. arXiv preprint arXiv:2605.10938, 2026.

Jiaxiu Jiang, Jingjing Ren, Wenbo Li, Bo Wang, Haoze Sun, Yijun Yang, Jianhui Liu, Yanbing Zhang, Shenghe Zheng, Yuan Zhang, Haoyang Huang, Nan Duan, and Wangmeng Zuo. TextLDM: Language Modeling with Continuous Latent Diffusion. arXiv preprint arXiv:2605.07748, 2026.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the Design Space of Diffusion-Based Generative Models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 26565–26577. Curran Associates, Inc., 2022.

Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis, 2021.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. RACE: Large-scale ReAding Comprehension Dataset From Examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 785–794, Copenhagen, Denmark, 2017. Association for Computational Linguistics.

Tianhong Li and Kaiming He. Back to Basics: Let Denoising Generative Models Denoise. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 36115–36125, 2026.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-LM Improves Controllable Text Generation. In *Advances in Neural Information Processing Systems*, volume 35, pp. 4328–4343. Curran Associates, Inc., 2022.

Chin-Yew Lin. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, 2004. Association for Computational Linguistics.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling, 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow, 2022.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete Diffusion Modeling by Estimating the Ratios of the Data Distribution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 32819–32848. PMLR, 2024.

Justin Lovelace, Varsha Kishore, Chao Wan, Eliot Shekhtman, and Kilian Q. Weinberger. Latent Diffusion for Language Generation. In *Advances in Neural Information Processing Systems*, volume 36, pp. 56998–57025. Curran Associates, Inc., 2023.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models. arXiv preprint arXiv:2211.01095, 2025.

Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent Consistency Models: Synthesizing High-Resolution Images with Few-Step Inference. arXiv preprint arXiv:2310.04378, 2023.

Viacheslav Meshchaninov, Egor Chimbulatov, Alexander Shabalin, Aleksandr Abramov, and Dmitry Vetrov. Cosmos: Compressed and Smooth Latent Space for Text Diffusion Modeling. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 14271–14299. Curran Associates, Inc., 2025.

Viacheslav Meshchaninov, Alexander Shabalin, Egor Chimbulatov, Nikita Gushchin, Ilya Koziev, Alexander Korotin, and Dmitry Vetrov. How to Train Your Latent Diffusion Language Model Jointly With the Latent Space. arXiv preprint arXiv:2605.07933, 2026.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, 2018. Association for Computational Linguistics.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A Corpus and Cloze Evaluation for Deeper Understanding of Commonsense Stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 839–849, San Diego, California, 2016. Association for Computational Linguistics.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807, Brussels, Belgium, 2018. Association for Computational Linguistics.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large Language Diffusion Models. arXiv preprint arXiv:2502.09992, 2025.

OpenAI. GPT-4 Technical Report. arXiv preprint arXiv:2303.08774v1, 2023.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. MAUVE: Measuring the Gap Between Neural Text and Human Text using Divergence Frontiers. In *Advances in Neural Information Processing Systems*, volume 34, pp. 4816–4828. Curran Associates, Inc., 2021.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. OpenAI technical report, 2019.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, 2016. Association for Computational Linguistics.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis With Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, 2022.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and Effective Masked Diffusion Language Models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 130136–130184. Curran Associates, Inc., 2024.

Subham Sekhar Sahoo, Justin Deschenaux, Aaron Gokaslan, Guanghan Wang, Justin T Chiu, and Volodymyr Kuleshov. The Diffusion Duality. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 52584–52619. PMLR, 2025.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, 2020.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense Reasoning about Social Interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, Hong Kong, China, 2019. Association for Computational Linguistics.

Alexander Shabalin, Viacheslav Meshchaninov, Egor Chimbulatov, Vladislav Lapikov, Roman Kim, Grigory Bartosh, Dmitry Molchanov, Sergey Markov, and Dmitry Vetrov. TEncDM: Understanding the Properties of the Diffusion Model in the Space of Language Model Encodings. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(23):25110–25118, 2025.

Noam Shazeer. GLU Variants Improve Transformer. arXiv preprint arXiv:2002.05202, 2020.

Junzhe Shen, Jieru Zhao, Ziwei He, and Zhouhan Lin. CoDAR: Continuous Diffusion Language Models are More Powerful Than You Think. arXiv preprint arXiv:2603.02547, 2026.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency Models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 32211–32252. PMLR, 2023.

Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, Yuwei Fu, Jing Su, Ge Zhang, Wenhao Huang, Mingxuan Wang, Lin Yan, Xiaoying Jia, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Yonghui Wu, and Hao Zhou. Seed Diffusion: A Large-Scale Diffusion Language Model with High-Speed Inference. arXiv preprint arXiv:2508.02193, 2025.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced Transformer with Rotary Position Embedding. arXiv preprint arXiv:2104.09864, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models. arXiv preprint arXiv:2302.13971, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Xinlong Wang, Xiaosong Zhang, Zhengxiong Luo, Quan Sun, Yufeng Cui, Jinsheng Wang, Fan Zhang, Yueze Wang, Zhen Li, Qiying Yu, Yingli Zhao, Yulong Ao, Xuebin Min, Tao Li, Boya Wu, Bo Zhao, Bowen Zhang, Liangdong Wang, Guang Liu, Zheqi He, Xi Yang, Jingjing Liu, Yonghua Lin, Tiejun Huang, and Zhongyuan Wang. Emu3: Next-Token Prediction is All You Need. arXiv preprint arXiv:2409.18869, 2024.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 Technical Report. arXiv preprint arXiv:2505.09388, 2025.

Ling Yang, Zixiang Zhang, Zhilong Zhang, Xingchao Liu, Minkai Xu, Wentao Zhang, Chenlin Meng, Stefano Ermon, and Bin Cui. Consistency Flow Matching: Defining Straight Flows with Velocity Consistency. arXiv preprint arXiv:2407.02398, 2024.

Zhihan Yang, Wei Guo, Shuibai Zhang, Subham Sekhar Sahoo, Yongxin Chen, Arash Vahdat, Morteza Mardani, and John Thickstun. Continuous Diffusion Scales Competitively with Discrete Diffusion for Language. arXiv preprint arXiv:2605.18530, 2026.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion Large Language Models. arXiv preprint arXiv:2508.15487, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics.

Biao Zhang and Rico Sennrich. Root Mean Square Layer Normalization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Tianyi Zhang, Chengcheng Liu, Jinwei Chen, Chun-Le Guo, Chongyi Li, Ming-Ming Cheng, Bo Li, and Peng-Tao Jiang. FlowConsist: Make Your Flow Consistent with Real Trajectory. arXiv preprint arXiv:2602.06346, 2026.

Siyan Zhao, Devaansh Gupta, Qinqing Zheng, and Aditya Grover. d1: Scaling Reasoning in Diffusion Large Language Models via Reinforcement Learning. arXiv preprint arXiv:2504.12216, 2025.

A Additional Method and Experimental Details

This appendix supplements the main text with implementation details, evaluation protocols, complete controlled-study results, and qualitative examples. It begins with the remaining method specifications, followed by the datasets, model configurations, training procedures, and evaluation settings, and concludes with the full quantitative results and generated samples.

A.1 Method and Implementation Details

A.1.1 Autoencoder Architecture and Reconstruction Objective

Algorithm 1 gives the full layer updates and prefix bounds for $N = \mathrm{round}(cL)$. The encoder input and decoder output projection share a trainable token embedding matrix $E \in \mathbb{R}^{|V|\times d}$. The operators Mask_x and Mask_z apply the training-time masking in Appendix A.1.2 and are identities at evaluation time. We apply RoPE (Su et al., 2023) to both sequences before attention to encode their positions.

Algorithm 1 Prefix-Aligned Autoencoding

Require: Tokens $w_{1:L}$, retention ratio c, encoder and decoder parameters
Ensure: Encoded latents z_{enc} and token logits $o_{1:L}$

$N \leftarrow \mathrm{round}(cL)$; $X \leftarrow P_{\mathrm{tok}}(\mathrm{Mask}_x(E[w_{1:L}]))$
$z^{(0)}_{\mathrm{enc},i} \leftarrow q$ for $i = 1, \ldots, N$
for $\ell = 0, \ldots, L_{\mathrm{enc}} - 1$ **do**
  For all $i = 1, \ldots, N$ in parallel:
    $\widetilde{z}^{(\ell)}_{\mathrm{enc},i} \leftarrow z^{(\ell)}_{\mathrm{enc},i} + \mathrm{MHA}^{(\ell)}_{\mathrm{enc}}(z^{(\ell)}_{\mathrm{enc},i}, [z^{(\ell)}_{\mathrm{enc},1:i}; X_{1:\lceil iL/N\rceil}])$
    $z^{(\ell+1)}_{\mathrm{enc},i} \leftarrow \widetilde{z}^{(\ell)}_{\mathrm{enc},i} + \mathrm{FFN}^{(\ell)}_{\mathrm{enc}}(\widetilde{z}^{(\ell)}_{\mathrm{enc},i})$
end for
$z_{\mathrm{enc}} \leftarrow P_{\mathrm{enc}}(\mathrm{RMSNorm}(z^{(L_{\mathrm{enc}})}_{\mathrm{enc}}))$
$Z \leftarrow P_{\mathrm{dec}}(\mathrm{Mask}_z(z_{\mathrm{enc}}))$
$h^{(0)}_j \leftarrow q_{\mathrm{dec}}$ for $j = 1, \ldots, L$
for $\ell = 0, \ldots, L_{\mathrm{dec}} - 1$ **do**
  For all $j = 1, \ldots, L$ in parallel:
    $\widetilde{h}^{(\ell)}_j \leftarrow h^{(\ell)}_j + \mathrm{MHA}^{(\ell)}_{\mathrm{dec}}(h^{(\ell)}_j, [h^{(\ell)}_{1:j}; Z_{1:\lfloor (j-1)N/L\rfloor+1}])$
    $h^{(\ell+1)}_j \leftarrow \widetilde{h}^{(\ell)}_j + \mathrm{FFN}^{(\ell)}_{\mathrm{dec}}(\widetilde{h}^{(\ell)}_j)$
end for
$o_j \leftarrow h^{(L_{\mathrm{dec}})}_j E^\top$ for $j = 1, \ldots, L$

Here P_{tok} maps token embeddings to the encoder hidden width H_{AE}, P_{enc} maps from H_{AE} to D, and P_{dec} maps the latent to the decoder input width. Only query states are updated across layers; the encoder token inputs X and decoder latent inputs Z remain fixed within each forward pass. In the latent-width study, we vary D through linear projections at the encoder output and decoder input, keeping the Transformer depths and hidden widths fixed.

We train the encoder and decoder jointly by matching the reconstructed logits to the original tokens with token-level cross-entropy:

$$\mathcal{L}_{\mathrm{AE}} = -\mathbb{E}_w \left[\frac{1}{L} \sum_{j=1}^{L} \log \mathrm{softmax}(o_j)_{w_j} \right]. \tag{7}$$

A.1.2 Autoencoder Training-Time Masking

We apply token-embedding dropout and latent dropout during autoencoder training (Meshchaninov et al., 2025). This subsection specifies these two masking operators used in our implementation. Using the trainable embedding matrix $E \in \mathbb{R}^{|V|\times d}$ defined in Sec. 3.2, token-embedding dropout independently zeros the input vector at each valid token position, encouraging the encoder to use surrounding context rather than rely on individual tokens. For $j \in \{1, \ldots, L\}$, let $m_j \sim \mathrm{Bernoulli}(p_x)$;

the masked encoder input is

$$\widetilde{E}[w_j] = (1 - m_j)E[w_j], \qquad p_x = 0.3. \tag{8}$$

After encoding, latent dropout independently masks each coordinate of $z_{\text{enc}} \in \mathbb{R}^{N \times D}$ before decoding, discouraging the decoder from relying on a small subset of latent features:

$$\widetilde{z}_{\text{enc}} = z_{\text{enc}} \odot r, \qquad r_{ik} \sim \text{Bernoulli}(1 - p_z), \quad i \in \{1, \ldots, N\},\ k \in \{1, \ldots, D\}, \qquad p_z = 0.6, \tag{9}$$

with no inverted-dropout rescaling.

A.1.3 Noisy-Input Projection

Each noisy latent is mapped to the Transformer hidden width H through two linear layers with intermediate width D_b:

$$W_{\text{in}} = W_{\text{up}} W_{\text{down}}, \qquad W_{\text{down}} \in \mathbb{R}^{D_b \times D}, \quad W_{\text{up}} \in \mathbb{R}^{H \times D_b}. \tag{10}$$

When $D_b < \min(D, H)$, the input projection has rank at most D_b. The bottleneck applies only to the noisy-input pathway; the output head retains the full clean-latent width D.

A.1.4 Noise-Level Sampling and High-Noise Mass

The tan-d mapping used in the main text follows a shifted cosine parameterization (Hoogeboom et al., 2023).

$$\sigma(\tau; d) = \frac{d \tan(\pi\tau/2)}{1 + d \tan(\pi\tau/2)}, \tag{11}$$

where larger values of d assign more training mass to highly corrupted states. Here $\sigma = t$ in the linear interpolant. The implementation samples $\tau \sim \mathcal{U}(10^{-3}, 1)$ and clips the resulting noise level to $[10^{-5}, 1 - 10^{-5}]$.

To compare different noise-level distributions, Sec. 4.2 uses the high-noise mass $m_{\sigma_\star} = \Pr(\sigma > \sigma_\star)$ at a threshold $\sigma_\star \in (0, 1)$. For the idealized mapping with $\tau \sim \mathcal{U}(0, 1)$, before the implementation cutoffs, the high-noise mass is

$$m_{\sigma_\star} = \Pr[\sigma(\tau; d) > \sigma_\star] = 1 - \frac{2}{\pi} \arctan\left(\frac{\sigma_\star}{d(1 - \sigma_\star)} \right). \tag{12}$$

For logit-normal sampling, $\ell \sim \mathcal{N}(\mu, s^2)$ and $\sigma = \text{sigmoid}(\ell)$. Since the sigmoid mapping is monotone, the corresponding high-noise mass is

$$m_{\sigma_\star} = \Pr[\sigma > \sigma_\star] = 1 - \Phi\left(\frac{\text{logit}(\sigma_\star) - \mu}{s} \right), \tag{13}$$

where Φ denotes the standard normal cumulative distribution function. Setting $\sigma_\star = 0.7$ gives the same $m_{0.7}$ coordinate used to compare the tan-d and logit-normal sweeps in Sec. 4.2.

A.1.5 Loss-Space Conversion for Clean-Latent Prediction

Under the linear interpolant used in flow matching (Lipman et al., 2023; Liu et al., 2022), Eq. (1) uses t as the interpolation coefficient. To match the notation of the controlled study, we denote this coefficient by σ. Setting $t = \sigma$ and $x_0 = z$, where z denotes the clean latent, gives the noisy latent and its path velocity as

$$x_\sigma = (1 - \sigma)z + \sigma\varepsilon, \qquad v = \varepsilon - z, \qquad \varepsilon \sim \mathcal{N}(0, I). \tag{14}$$

If the model predicts the clean latent $\hat{z}_\theta(x_\sigma, \sigma)$, the corresponding velocity prediction is

$$\hat{v}_\theta(x_\sigma, \sigma) = \frac{x_\sigma - \hat{z}_\theta(x_\sigma, \sigma)}{\sigma}, \qquad \|\hat{v}_\theta - v\|_2^2 = \frac{1}{\sigma^2} \|\hat{z}_\theta(x_\sigma, \sigma) - z\|_2^2. \tag{15}$$

Therefore, evaluating a clean-latent prediction in velocity space multiplies its clean-latent error by $1/\sigma^2$, assigning greater weight to samples at lower noise levels. This is the loss-space effect examined in Sec. 4.2.

A.1.6 Self-Trajectory Consistency Implementation

During STC training, we obtain $t' = \mathrm{Prev}_S(t)$ by taking a step of size $1/S$ toward lower noise in the tan-d schedule coordinate. For the tan-d mapping in Eq. (11),

$$\tau = \frac{2}{\pi}\arctan\left(\frac{t}{d(1-t)}\right), \qquad t' = \sigma\big(\max\{\tau - 1/S, 10^{-3}\}; d\big)\,. \tag{16}$$

The resulting t' is clipped to $[10^{-5}, 1-10^{-5}]$. The state update and consistency loss follow Eqs. (5) and (6). For conditional training, visible prompt positions are restored to their clean latents after the update, using the same conditioning-dropout mask. When self-conditioning is enabled, the EMA input is the detached current prediction, including updates without a draft estimate for the current model.

The consistency weight is increased linearly from zero to its final value λ_{ct} over the first T_{warm} optimization steps. Unless self-trajectory consistency is explicitly disabled, the reported runs use

$$\lambda_{\mathrm{ct}} = 1.0, \qquad S = 16. \tag{17}$$

We set $T_{\mathrm{warm}} = 10{,}000$ optimization steps for AURORA-LM-S and $T_{\mathrm{warm}} = 5{,}000$ optimization steps for AURORA-LM-L.

A.1.7 Guidance

At each solver step, the diffusion model predicts a clean estimate of the current latent block. Guidance modifies this estimate by combining two model predictions before updating the noisy state. For unconditional generation, the clean estimate from the preceding solver step serves as the conditioning signal for SC-CFG. When this signal is available, the model produces one prediction with self-conditioning and one with the self-conditioning input omitted, denoted by $\hat{\alpha}^{(b)}_{\mathrm{sc}}$ and $\hat{\alpha}^{(b)}_{\text{no-sc}}$, respectively. SC-CFG combines them as:

$$\hat{\alpha}^{(b)}_{\text{SC-CFG}} = \hat{\alpha}^{(b)}_{\text{no-sc}} + w_{\mathrm{sc}}\left(\hat{\alpha}^{(b)}_{\mathrm{sc}} - \hat{\alpha}^{(b)}_{\text{no-sc}}\right). \tag{18}$$

Here, $w_{\mathrm{sc}} = 1$ recovers the self-conditioned estimate, while larger values strengthen the direction favored by self-conditioning, providing inference-time control over the trade-off between generation quality and diversity.

For prompt-conditioned generation, standard classifier-free guidance (Ho & Salimans, 2022) uses one prediction with the prompt prefix and one without it:

$$\hat{\alpha}^{(b)}_{\mathrm{CFG}} = \hat{\alpha}^{(b)}_{\mathrm{uncond}} + w\left(\hat{\alpha}^{(b)}_{\mathrm{cond}} - \hat{\alpha}^{(b)}_{\mathrm{uncond}}\right). \tag{19}$$

In this case, $w = 1$ recovers the prompt-conditioned estimate, while larger values strengthen the effect of the prompt. Both guidance rules form linear combinations of clean-latent predictions at inference time and do not alter the training objective. We study the SC-CFG scale in Appendix A.3.7.

A.2 Experimental Protocol and System Configuration

This section specifies the data preprocessing, model architecture, training configuration, and generation and evaluation protocols used throughout the experiments. It also documents baseline reproduction and public-benchmark evaluation. Complete inference sweeps are reported in Appendix A.3.7 and Appendix A.3.8.

A.2.1 Training-Data Preprocessing

For both the controlled design analysis and the system-level comparison, documents are drawn from OpenWebText (Gokaslan & Cohen, 2019). The controlled analysis uses the Qwen3 tokenizer with a maximum sequence length of 128 tokens (OWT128), whereas the system-level comparison uses the GPT-2 BPE tokenizer and packs documents into sequences of 1,024 tokens. For a fair system-level comparison, we follow the OpenWebText data split and preprocessing protocol used

by Duo (Sahoo et al., 2025), including fixed-length wrap packing. Each sequence begins and ends with an EOS token, and adjacent documents within a packed sequence are separated by EOS.

For AURORA-LM-L, we construct a 294.7B-token training mixture from diverse English-language resources, including public web, books, academic and mathematical text, as well as synthetically generated text. After source-level weighting and packing, the mixture comprises 161.3B web tokens (54.7%), 50.3B tokens from books and encyclopedia (17.1%), and 83.1B tokens of academic, mathematical, and knowledge text (28.2%). These counts describe the constructed corpus rather than the total token exposures accumulated over optimization. The model uses Qwen3 tokenization, and the resulting documents are packed into 512-token sequences.

A.2.2 Model Architecture

Table 3: **Architectures of AURORA-LM-S and AURORA-LM-L.** AURORA-LM-S is used for the system-level comparisons, whereas AURORA-LM-L is used for the scaling evaluation. Model-size labels refer to the block-causal denoiser and exclude the autoencoder.

Component	AURORA-LM-S (130M)	AURORA-LM-L (1B)
Query-based Encoder-Decoder		
Encoder layers / decoder layers	6 / 6	6 / 6
Hidden size / heads / head dim	768 / 12 / 64	1024 / 16 / 64
Feed-forward intermediate	3072	4096
Latent channel D	768	1024
Latent length N	1024 (c=1)	512 (c=1)
Token-embedding initialization	GPT-2 small	Qwen3-0.6B
Encoder parameters	49.6M	88.1M
Decoder parameters	88.2M, including tied output projection	243.7M, including tied output projection
Block-causal denoiser		
Layers	12	24
Hidden size / heads / head dim	768 / 12 / 64	1536 / 12 / 128
Feed-forward intermediate	3072	6144
Noisy-input bottleneck D_b	128	128
Block size Q	16	16
Positional encoding	RoPE (θ=10,000, max_pos 1024)	RoPE (θ=10,000, max_pos 512)
Attention mode	Two-stream: clean prefix to noised current block	Two-stream: clean prefix to noised current block
Denoiser parameters	130M	1.01B
Total system parameters	267.8M	≈1.34B

The token-embedding tables are initialized from the listed pretrained models and updated during autoencoder training. The trained autoencoder, including its tied output embedding, is then frozen before denoiser training.

The block-causal denoiser is implemented as a Transformer (Vaswani et al., 2017) using RMSNorm (Zhang & Sennrich, 2019), SwiGLU feed-forward layers (Shazeer, 2020), RoPE positional encoding (Su et al., 2023), and per-head QK normalization (Henry et al., 2020). Timestep conditioning uses sinusoidal embeddings (Ho et al., 2020) projected through a two-layer MLP and injected additively into each Transformer layer.

A.2.3 Training Configuration

Table 4 summarizes the training configurations for AURORA-LM-S and AURORA-LM-L. The per-channel mean and standard deviation are estimated from valid encoder outputs on held-out data and kept fixed during diffusion training and generation. Padding positions are masked, and conditioning dropout hides the dropped context from noisy-stream queries.

Table 4: **Training configurations for AURORA-LM-S and AURORA-LM-L in Table 3.** The underlying objectives are defined in §3 and Appendix A.1. Optimization steps, warmup durations, and EMA frequencies refer to parameter updates; global batch sizes are reported after gradient accumulation. Training compute is estimated using $3ND$ accounting, with one multiply–accumulate counted as one operation.

Setting	AURORA-LM-S (130M)	AURORA-LM-L (1B)
Shared (Autoencoder + Denoiser)		
Optimizer	AdamW, β=$(0.9, 0.98)$	AdamW, β=$(0.9, 0.98)$
Gradient clipping	1.0	1.0
Mixed precision	bf16	bf16
Random seed	42	42
Autoencoder training		
Optimization steps	100,000	150,000
Global batch size	1024	2048
Weight decay	10^{-2}	10^{-2}
Learning rate	4×10^{-4}	2×10^{-4}
Token-embedding dropout p_x	0.3	0.3
Latent feature dropout p_z	0.6	0.6
Training compute	$\approx$ 43 EFLOPs	$\approx$ 157 EFLOPs
Denoiser training		
Optimization steps	250,000 (OWT); 300,000 (XSum)	590,000
Global batch size	1024 (OWT); 128 (XSum)	1024
ε	10^{-6}	10^{-6}
Learning rate	4×10^{-4}	Peak 2×10^{-4}, minimum 1×10^{-6}
Weight decay	10^{-5}	10^{-5}
Warmup	1000 steps	1000 steps
EMA	0.9999, every optimization step	0.9999, every optimization step
Parameterization	Flow matching, $x_\sigma = (1-\sigma)z+\sigma\varepsilon$	Flow matching, $x_\sigma = (1-\sigma)z+\sigma\varepsilon$
Prediction target	x_0 (clean latent)	x_0 (clean latent)
Loss space	x_0-space	x_0-space
Time sampling	tan-d, shift 7.0	tan-d, shift 7.0
Self-conditioning	p_{sc}=0.6	p_{sc}=0.5
Self-trajectory consistency	λ_{ct}=1.0, S=16	λ_{ct}=1.0, S=16
Consistency warmup	10,000 steps	5,000 steps
Latent standardization	Per-channel, held-out encoder statistics	Per-channel, held-out encoder statistics
Instruction fine-tuning	None	None
Training compute	$\approx$ 155 EFLOPs (OWT); $\approx$ 23 EFLOPs (XSUM)	$\approx$ 1,406 EFLOPs

A.2.4 Controlled OWT128 Studies

Table 5 summarizes the default configuration for the controlled OWT128 studies in Sec. 4.2.

A.2.5 Generation and Evaluation Protocols

The system-level OpenWebText comparison (Sec. 4.1) uses 64-step SDE-DPM++ sampling with self-conditioning classifier-free guidance (SC-CFG) at $w_{\text{sc}} = 4$. The XSum conditional comparison (Sec. 4.1) uses 32-step ODE sampling with classifier-free guidance at scale $w = 2.0$.

The public benchmark uses 16 denoising steps, guidance scale 3.0, and at most 32 newly generated tokens. AURORA-LM-L benchmark evaluation uses a fixed base sampling seed of 256. For AURORA-LM on XSum and the public benchmark, CFG is applied only in the high-noise window $\sigma \in [0.9, 1.0]$.

Cola-DLM is evaluated using its publicly released checkpoint and inference implementation (Guo et al., 2026). We retain its release defaults: a 16-step Euler sampler, CFG scale 7.0, and per-sample inference-noise base seed 66. Both systems are evaluated on the same benchmark instances using the shared prompt, truncation, and answer-matching protocol in Sec. A.2.8; sampling configurations

Table 5: **Default configuration for the controlled OWT128 studies.** Unless otherwise specified, settings follow this configuration. Sweep-specific widths, schedules, block sizes, and sampling budgets are given with the full results in Appendix A.3.

Setting	Default value
Architecture and latent configuration	
Maximum token sequence length	128
Encoder layers / decoder layers	6 / 6
AE hidden / feed-forward width	1024 / 4096
Token-embedding initialization	Qwen3-0.6B
Denoiser layers	12
Denoiser hidden / heads / head dim	1024 / 16 / 64
Denoiser feed-forward width	4096
Latent width D / retention ratio c	1024 / 1
Noisy-input width D_b / block size Q	128 / 16
Autoencoder and denoiser training	
Optimizer (both stages)	AdamW, $\beta = (0.9, 0.98)$
Optimization steps (AE / denoiser)	50,000 / 50,000
Global batch size (both stages)	1024
Learning rate (both stages)	4×10^{-4}
Linear warmup (both stages)	1000 optimization steps
Learning-rate schedule	Constant after warmup
Weight decay (AE / denoiser)	10^{-2} / 10^{-5}
Gradient clipping (both stages)	1.0
Noise allocation	tan-d, $d = 5$
Prediction target / loss space	x_0 / x_0
Evaluation	
Evaluation sampler	32-step ODE, no guidance
Generation seeds / samples per seed	1–5 / 1,000

remain model-specific. We do not retrain Cola-DLM, so its training configuration is not treated as matched to that of AURORA-LM.

For prefix-conditioned generation (XSum and the public benchmark), we first encode the prompt into a clean latent prefix and use its complete blocks as causal context for subsequent blockwise generation, following Guo et al. (2026). If the prefix terminates within a block, our implementation uses clean repainting: the known prefix positions in this boundary block are fixed to their encoder-produced latents throughout denoising, while the remaining positions are generated.

A.2.6 Reproduction Protocol for Baselines

For the OpenWebText unconditional-generation comparison (Sec. 4.1), we evaluate released checkpoints from the Duo (Sahoo et al., 2025) and ELF (Hu et al., 2026) repositories. The ELF MAUVE evaluation uses a separate reference-construction and length-alignment protocol, described below. For the XSum conditional-generation comparison (Sec. 4.1), the reported metrics are taken directly from ELF (Hu et al., 2026). We do not retrain any baseline end to end; results obtained under our evaluation harness differ from the original publications only in the evaluation harness (sampling code, scorer, and metric implementation), not in the underlying model weights or training recipe.

AURORA-LM-S uses the same hidden size (768) and depth (12 layers) as the system-level baselines, but its block-causal denoiser contains 130M parameters, compared with approximately 92M for the discrete-diffusion backbones. This difference primarily reflects its SwiGLU feed-forward layers, which retain the 3,072-dimensional intermediate width while adding a gating projection, together with the timestep-conditioning pathway and the per-layer self-conditioning projections. These backbone counts exclude token embeddings, vocabulary heads, and AURORA-LM's frozen autoencoder.

Table 6: **Model architecture and evaluation configurations across baselines.** The 105M parameters reported for ELF-B in Table 1 include its backbone and LM head. These exclude the frozen T5-small input encoder (35M), which is used only during training and not at inference.

	AR	MDLM	SEDD	DUO	DUO-distilled	ELF-B
Hidden size	768	768	768	768	768	768
Num layers	12	12	12	12	12	12
Num heads	12	12	12	12	12	12
Head dim	64	64	64	64	64	64
Intermediate size	3072	3072	3072	3072	3072	3072
Vocab size	50257	50257	50257	50257	50257	32128
Tokenizer	GPT-2 BPE	GPT-2 BPE	GPT-2 BPE	GPT-2 BPE	GPT-2 BPE	T5 (SentencePiece)
Sequence length	1024	1024	1024	1024	1024	1024
Sampling steps	1024 tokens	1024	1024	1024	32	32
Sampling method	AR (Gumbel-max)	Ancestral	Analytic	Ancestral	Ancestral	SDE
Backbone params	85.00M	92.13M	92.13M	92.13M	92.13M	88M
LM head params	38.65M	38.85M	38.85M	38.85M	38.85M	17M
Vocab embedding params	38.60M	38.60M	38.60M	38.60M	38.60M	–
Total params	**162.25M**	**169.58M**	**169.58M**	**169.58M**	**169.58M**	**105M**

A.2.7 MAUVE Evaluation Protocol

The MAUVE scores in Table 1(a) use held-out OpenWebText references constructed with GPT-2 wrap preprocessing. We evaluate 1,000 generated and 1,000 reference texts per seed, truncating both to at most 1,024 GPT-2 tokens before extracting GPT-2 Large features. Scores are averaged over five generation seeds.

ELF-B MAUVE reevaluation. We sample references from the OpenWebText-T5 dataset released with ELF and decode them with T5 to match ELF-B's generated text. ELF-B's generations and these references differ in length, which can affect MAUVE (Shabalin et al., 2025). To control this difference, we exclude texts shorter than 768 GPT-2 tokens and score the first 768 tokens of 1,000 generations and 1,000 references per seed. Across five seeds, this yields MAUVE 0.768 ± 0.062. We report the result separately because the reference data and truncation differ from those used for Table 1(a); the scores are not directly comparable.

A.2.8 Few-Shot Benchmark Protocol

MMLU. A multiple-choice benchmark covering humanities, STEM, social sciences, and professional domains (Hendrycks et al., 2021), used to assess broad factual knowledge and reasoning.

HellaSwag. A multiple-choice benchmark for grounded commonsense reasoning, requiring selection of the most plausible sentence continuation from adversarially constructed candidates (Zellers et al., 2019).

ARC-Challenge. A multiple-choice benchmark of grade-school science questions selected to be difficult for retrieval- and word-co-occurrence-based baselines, requiring scientific knowledge and reasoning (Clark et al., 2018).

WinoGrande. A large-scale, bias-reduced pronoun-resolution benchmark inspired by the Winograd Schema Challenge, used to assess commonsense reasoning (Sakaguchi et al., 2020).

OpenBookQA. A multiple-choice benchmark combining core science facts with broader commonsense knowledge, often requiring multi-hop reasoning (Mihaylov et al., 2018).

SIQA. A social commonsense reasoning benchmark requiring selection of the most plausible response to social situations and intentions (Sap et al., 2019).

StoryCloze. A story understanding benchmark requiring selection of the most plausible ending for a four-sentence story context, evaluating narrative coherence and causal reasoning (Mostafazadeh et al., 2016).

RACE. A reading comprehension benchmark from English examinations, requiring passage understanding and inference beyond simple span extraction (Lai et al., 2017).

SQuAD. A reading-comprehension benchmark in which answers are spans in a given passage (Rajpurkar et al., 2016). In our evaluation, the model generates a short answer conditioned on the passage and question; we report normalized exact match.

For each benchmark task, we use the standard public evaluation split: the official test split when available and the official validation split otherwise. We construct a fixed candidate subset by shuffling the corresponding split with the default subset-selection seed of 42 and retaining up to 1,000 examples. To ensure matched evaluation, we retain only candidates that satisfy the context-budget requirements of both systems under the shared prompting and generation setup. The resulting fixed task sets are then used unchanged to evaluate both AURORA-LM-L and Cola-DLM.

We use a common prompt-construction, output-truncation, and answer-matching procedure throughout the benchmark evaluation. Tasks that support in-context demonstrations use two-shot prompts. In multiple-choice demonstrations, answers are written as the full option text rather than the option letter, following the generative evaluation protocol used by Cola-DLM (Guo et al., 2026). For multiple-choice tasks, the generated response is truncated at the first newline, normalized, and mapped to a candidate option; the shared answer matcher accepts either an explicit option letter or the corresponding option text. For SQuAD, we apply the same first-newline truncation and standard SQuAD normalization to both the generated answer and each reference alias: lowercasing, punctuation and article removal, and whitespace normalization. We report exact match against the best-matching reference alias. The exact demonstrations used in these prompts are listed in Appendix A.6 (Table 17).

Table 7: **Benchmark evaluation protocol.** All tasks use two-shot prompts with textual answer completions. For multiple-choice tasks, the scoring parser accepts either the option text or an explicit option letter.

Task	Shots	Accepted output	Scoring
HellaSwag	2	Option text (or letter)	Accuracy
MMLU	2	Option text (or letter)	Accuracy
OpenBookQA	2	Option text (or letter)	Accuracy
RACE	2	Option text (or letter)	Accuracy
SIQA	2	Option text (or letter)	Accuracy
SQuAD	2	Short answer text	Normalized exact match (EM)
StoryCloze	2	Option text (or letter)	Accuracy
ARC-Challenge	2	Option text (or letter)	Accuracy
WinoGrande	2	Option text (or letter)	Accuracy

A.3 Additional Numerical Results

The controlled OWT128 analyses in Sec. 4.2 use five generation seeds (1–5), with 1,000 samples per run. Tables report mean ± standard deviation in MAUVE against held-out OpenWebText references; shaded cells mark the best MAUVE within each controlled comparison. The final two subsections report the system-level OpenWebText and XSum inference-protocol sweeps.

A.3.1 Latent Width and Noise Allocation

Table 8: **Full channel-width and tan-d schedule matrix.** Each cell reports mean MAUVE ± standard deviation over evaluation seeds. Bold marks the best schedule for each latent width.

Latent channel D	$d = 1$	$d = 3$	$d = 5$	$d = 7$
128	0.345±0.027	0.578±0.025	**0.594±0.035**	0.509±0.039
256	0.230±0.012	0.616±0.025	0.604±0.031	**0.640±0.033**
512	0.236±0.018	0.637±0.022	**0.693±0.011**	0.667±0.036
1024	0.229±0.023	0.797±0.032	0.815±0.012	**0.839±0.024**

A.3.2 Noisy-Input Bottleneck

Table 9: **Detailed noisy-input bottleneck sweep for a full-width clean target with $D = 1024$.** Each step-specific entry reports mean MAUVE $\pm$ standard deviation over evaluation seeds. The final column averages the three step-specific means and is the statistic shown in Fig. 4(c).

D_b	16 steps	32 steps	64 steps	Mean
32	0.793±0.019	0.795±0.009	0.771±0.025	0.786
128	0.774±0.013	0.815±0.012	0.835±0.027	**0.808**
256	0.728±0.025	0.797±0.046	0.754±0.023	0.760
512	0.706±0.043	0.779±0.023	0.759±0.047	0.748
1024	0.757±0.017	0.807±0.033	0.807±0.014	0.790

A.3.3 Noise Allocation

Figure 6 visualizes the training-time noise distributions used in the schedule ablation. Table 10 reports the corresponding high-noise mass and generation quality.

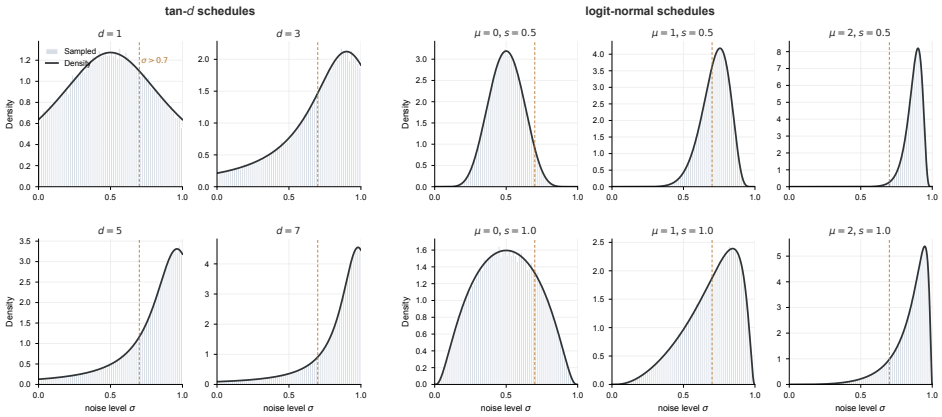

Figure 6: **Training-time noise distributions used in the schedule ablation.** Each panel shows the density over the flow-matching noise level $\sigma \in [0, 1]$. The dashed reference marks the high-noise region $\sigma > 0.7$ used in Fig. 5(a).

Table 10: **Noise-allocation sweep underlying Fig. 5(a).** The high-noise mass is $m_{0.7} = \Pr(\sigma > 0.7)$; MAUVE is reported as mean $\pm$ standard deviation over evaluation seeds.

Schedule family	Parameter	$m_{0.7}$	MAUVE ↑
tan-d	$d = 1$	0.258	0.229±0.023
tan-d	$d = 3$	0.579	0.797±0.032
tan-d	$d = 5$	0.722	0.815±0.012
tan-d	$d = 7$	0.795	**0.839±0.024**
logit-normal, $s = 0.5$	$\mu = 0$	0.045	0.010±0.001
logit-normal, $s = 0.5$	$\mu = 1$	0.620	0.111±0.011
logit-normal, $s = 0.5$	$\mu = 2$	0.989	0.828±0.017
logit-normal, $s = 1.0$	$\mu = 0$	0.198	0.022±0.003
logit-normal, $s = 1.0$	$\mu = 1$	0.561	0.586±0.051
logit-normal, $s = 1.0$	$\mu = 2$	0.875	0.833±0.014

A.3.4 Prediction Target and Loss Space

Table 11 reports all four prediction-target and loss-space combinations at two noisy-input widths. The corresponding loss-space conversion is given in Appendix A.1.5.

Table 11: **Target and loss-space analysis for a full-width clean target with $D = 1024$.** Direct clean-latent regression yields the highest MAUVE under both noisy-input widths.

Target	Loss	Noisy-input width	
		$D_b = 128$	$D_b = 1024$
x_0	x_0	$\mathbf{0.815 \pm 0.012}$	$\mathbf{0.807 \pm 0.033}$
	v	0.059 ± 0.009	0.017 ± 0.002
v	x_0	0.729 ± 0.023	0.325 ± 0.021
	v	0.653 ± 0.034	0.301 ± 0.016

A.3.5 Self-Trajectory Consistency

Table 12: **Self-trajectory consistency across denoising-step budgets.** Numerical results for Fig. 5(c), reported as mean MAUVE $\pm$ standard deviation over generation seeds.

Steps	Consistency off	Consistency on
8	0.116 ± 0.006	0.672 ± 0.038
16	0.310 ± 0.039	0.786 ± 0.034
32	0.589 ± 0.062	0.815 ± 0.012
64	0.660 ± 0.023	$\mathbf{0.832 \pm 0.014}$
128	$\mathbf{0.757 \pm 0.029}$	0.825 ± 0.024
256	0.724 ± 0.022	0.797 ± 0.014

A.3.6 Additional Sequence-Compression and Block-Size Results

Table 13: **Additional sequence-compression results.** The latent channel width is fixed at $D = 1024$ and the noisy-input bottleneck at $D_b = 128$.

Latent positions retained	MAUVE ↑
100%	$\mathbf{0.815 \pm 0.012}$
90%	0.807 ± 0.022
80%	0.726 ± 0.020
70%	0.671 ± 0.063

Table 14: **Additional block-size results.** Separately trained models are evaluated with 32-step ODE sampling and no guidance. Values are mean MAUVE ± standard deviation over generation seeds.

Q	**MAUVE ↑**
4	**0.909±0.012**
8	0.857±0.011
16	0.815±0.012
32	0.681±0.023
64	0.631±0.027

We use $Q = 16$ to balance left-to-right conditioning with within-block parallelism: on OWT128, it requires eight sequential block-generation stages, versus 32 at $Q = 4$.

A.3.7 System-Level Inference-Protocol Selection

Table 15 reports the complete inference-protocol sweep underlying the system-level OpenWebText comparison. At the fixed 250k checkpoint, we vary the denoising-step count, solver, and SC-CFG scale. Each configuration generates 1,000 samples for each of five evaluation seeds, and entries report the mean ± standard deviation. The corresponding architecture and generation configurations for the system-level baselines are listed in Table 6.

Within a fixed solver and step budget, increasing SC-CFG generally lowers Gen-PPL until the improvement saturates. Its effect on MAUVE is non-monotonic and depends on both the solver and step budget. We select the 64-step SDE-DPM++ configuration with $w_{\text{sc}} = 4$, which attains the lowest mean Gen-PPL in the sweep while retaining a MAUVE of 0.890.

Table 15: **Complete inference-protocol sweep for AURORA-LM-S at the 250k checkpoint under the OpenWebText free-generation protocol.** Results are grouped by denoising-step count so that the two solvers can be compared at the same SC-CFG scale. Each entry reports the mean ± standard deviation over evaluation seeds 1–5, with 1,000 samples per seed. The shaded cells mark the selected system-level configuration.

SC-CFG	**Euler**			**SDE-DPM++**		
	Gen-PPL ↓	**Entropy ↑**	**MAUVE ↑**	**Gen-PPL ↓**	**Entropy ↑**	**MAUVE ↑**
16 denoising steps						
1	72.72 ± 0.44	5.385 ± 0.002	0.8854 ± 0.0240	98.36 ± 1.36	5.449 ± 0.009	0.8481 ± 0.0271
2	60.57 ± 0.35	5.365 ± 0.001	0.9036 ± 0.0156	70.11 ± 0.24	5.412 ± 0.007	0.8888 ± 0.0212
3	57.27 ± 0.12	5.368 ± 0.003	0.9083 ± 0.0114	61.43 ± 0.29	5.396 ± 0.003	0.9214 ± 0.0211
4	54.72 ± 0.60	5.367 ± 0.007	0.9199 ± 0.0092	54.86 ± 0.44	5.361 ± 0.004	0.9199 ± 0.0120
5	54.92 ± 0.47	5.374 ± 0.004	0.9149 ± 0.0184	54.67 ± 0.52	5.380 ± 0.005	0.8952 ± 0.0215
32 denoising steps						
1	50.50 ± 0.33	5.323 ± 0.005	0.8930 ± 0.0228	62.01 ± 0.23	5.406 ± 0.002	0.9232 ± 0.0147
2	41.58 ± 0.42	5.309 ± 0.006	0.9271 ± 0.0072	41.53 ± 0.53	5.347 ± 0.009	0.9482 ± 0.0143
3	38.93 ± 0.26	5.314 ± 0.004	0.9440 ± 0.0120	35.44 ± 0.50	5.310 ± 0.007	$\mathbf{0.9516 \pm 0.0055}$
4	37.65 ± 0.41	5.316 ± 0.004	0.9344 ± 0.0178	31.63 ± 0.33	5.292 ± 0.004	0.9250 ± 0.0177
5	37.09 ± 0.19	5.312 ± 0.006	0.9225 ± 0.0103	31.95 ± 0.13	5.311 ± 0.005	0.9205 ± 0.0173
64 denoising steps						
1	42.55 ± 0.22	5.326 ± 0.002	0.9258 ± 0.0191	45.92 ± 0.38	5.391 ± 0.005	0.9407 ± 0.0131
2	34.01 ± 0.12	5.294 ± 0.004	0.9290 ± 0.0126	30.09 ± 0.18	5.310 ± 0.007	0.9242 ± 0.0119
3	30.65 ± 0.32	5.271 ± 0.006	0.9358 ± 0.0106	24.33 ± 0.21	5.243 ± 0.008	0.9016 ± 0.0160
4	29.30 ± 0.24	5.275 ± 0.005	0.9176 ± 0.0197	$\mathbf{23.56 \pm 0.30}$	5.241 ± 0.007	0.8896 ± 0.0222
5	29.43 ± 0.22	5.268 ± 0.006	0.9362 ± 0.0063	25.26 ± 0.10	5.247 ± 0.004	0.9028 ± 0.0075

A.3.8 XSum Inference-Protocol Selection

We select the classifier-free guidance scale and ODE step count for the XSum comparison using the 300k AURORA-LM-S checkpoint. The guidance sweep fixes the sampler at 32 ODE steps, while the step-count sweep fixes the guidance scale at $w = 2.0$. Each setting is evaluated with five seeds (1–5). Table 16 reports the mean and standard deviation across seeds for both sweeps. Here $w = 1.0$ uses the conditional prediction without additional classifier-free guidance.

Moderate guidance improves all three ROUGE metrics over $w = 1.0$. A scale of $w = 2.0$ gives the highest mean ROUGE-1 and ROUGE-L, while $w = 1.5$ gives a marginally higher ROUGE-2. Increasing the scale beyond this range degrades all three metrics. With $w = 2.0$, 32 ODE steps give the highest mean ROUGE-1 and ROUGE-L, while matching 64 steps in ROUGE-2 after rounding. Additional steps do not improve the overall ROUGE scores. We therefore use 32-step ODE sampling with $w = 2.0$ for the XSum comparison.

Table 16: **XSum inference-protocol selection for AURORA-LM-S.** Both sweeps report mean ± standard deviation over five sampling seeds. Shaded rows denote the selected configuration.

(a) Guidance scale at 32 ODE steps

w	ROUGE-1 ↑	ROUGE-2 ↑	ROUGE-L ↑
1.0	34.96±0.34	12.28±0.13	26.76±0.30
1.5	36.42±0.20	**13.47±0.24**	28.54±0.25
2.0	**36.56±0.15**	13.43±0.29	**28.86±0.18**
2.5	35.87±0.25	12.78±0.26	28.37±0.21
3.0	34.16±0.19	11.56±0.15	26.90±0.12

(b) ODE steps at $w = 2.0$

Steps	ROUGE-1 ↑	ROUGE-2 ↑	ROUGE-L ↑
16	36.37±0.22	13.15±0.26	28.58±0.24
32	**36.56±0.15**	**13.43±0.29**	**28.86±0.18**
64	36.45±0.14	**13.43±0.26**	28.75±0.14
128	36.32±0.17	13.31±0.26	28.59±0.19

A.4 Qualitative OWT Examples

To give a concrete view of AURORA-LM's unconditional generation behavior, we include two examples from the OWT1024 free-generation evaluation. Gen-PPL is computed for each example with the fixed GPT-2 Large scorer; Entropy is its token-unigram entropy over GPT-2 BPE tokens. The main quantitative comparison uses corpus-level results in Table 1(a).

Example 1 PPL: 24.625 Entropy: 5.467

Generation.

<|endoftext|> When talking about open source, there's a few options at play: open access to an existing project, give to existing members of the community a chance to continue what they've loved, or use it to develop code and features used by others.

Of course, there have been a litany of different options: for instance, in the past, Ruby contributors could just write Ruby code, then publish it, on the forums, include the source code, then use it for others to improve their code if they like. Well, this sounds logical to me. It would open up the community at the opportunity new people could use and improve on their original code without having to go through all of the large hoops that come with trying to provide good quality code.

Unfortunately, open source seems to be a possibility, so here comes Direct Air, a project run by a guy named Justin Vipme. He's the most devoted author of what looks like an entirely free, wide-used code base, without any permission.

Two years ago, Justin sent me an email I wanted to give you at least a overview of it. The first part I think here brought me an immediate interest in some module creation. Yes, you see, module creation—also called "backCreating"—is the complex process that takes place at any time in your application. Basically, working between the users and the developers is separate from working on your own system. On the other hand, the first thing I wish I could think of doing with Direct Air would be doing a bug report to the bug delivery team and then release a solution to make sure the project works right from the ground up.

So that's what surprised me at the time. You'll need to speak to some open source author running anything other than Ruby in order to make a bug report:

https://github.com/merick-vipme/backCreating.im

This involved taking a tip of each module that went wrong and trying to pin down how they were fixed. In many ways, the project actually helped make the process run faster and more efficient than many people had hoped. And it also made it much easier to developers to issue a bug report and make sure the module works as intended. Plus, a combination of each module that allows them to automatically communicate with the developers involved made it easy for

them to report an issue in their own system more readily and efficiently, rather than relying on a message engine.

You can see this happening in the make-up of the bug report below, and when you do, you should get a commit line that shows just which of the modules that went wrong was fixed.

As you can see, Vipme was dealing with 75% of bugs, which makes it easier and faster for developers to check if they're worth patches. And to keep in mind, this doesn't mean that somebody has tried to open a module or that somebody has accidentally written your package. It does mean that reaching out to others will help ensure that bugs aren't being fixed very easily.

The coolest part is that the bug reports

<|endoftext|>

Example 2 PPL: 22.00 Entropy: 5.358

Generation.

<|endoftext|> mediators have to carve out a compromise that is less likely than most. They will agree on strong demands as early as Monday or early Friday unless external demands are met, government officials said.

They will also agree on rights to limiting refugee arrivals in Belgium and stopping people from trying to travel across the Atlantic to countries including Sweden and France.

The EU is close to setting up talks with Russia, paving the way for similar talks with the UK, France and Italy in July and September of a need for more sharing. Those talks have already started in Geneva this week and in London.

George Murphy, the head of the European Commission, said that Britain should be "perceived" with an agreement on immigration, security and criminal justice on Wednesday.

But he said the crisis had not developed overnight. Europe should also look at the case-by-case role of external mediators in securing a solution.

Speaking from the EU headquarters, he acknowledged that the global migrant crisis is complex and said it was too early to resolve it.

He said the UK should not waste the opportunity of persuading all EU nations to introduce new measures, such as cutting access to shelters, in order to prevent Islamic State fighters from crossing the border.

He also suggested it should set up central migrant crisis directors' office in Belgium with experts on security, criminal justice and democracy as well as human rights.

EU officials indicated Thursday that representatives from would have to co-operate with 12 member states in order to come up with a strategy to meet their demands.

The number of asylum claims has more than doubled in the past two years and doubled in 2016, with more than 6,450 people now entering Europe after a decade of obstacles. There have been complaints about ticket queues at the border too long and locals say tourists are being denied housing.

David Cameron: 'We need to make sure not a single migrant comes in safely'

There has been an unprecedented surge of refugees coming to make it to Europe over the past year. There are between 9,000 and 5,000 a day a day from Iraq, Syria, Yemen, Yemen and Somalia. More than 5 million people live on the EU's Southern border which provides a major hub for smuggling and support Islamic State fighters in Syria and Somalia - from Iraq and Syria.

There have also been an unprecedented number of refugees arriving from Turkey over the past year which are about 268 refugees a day, what Cameron termed "the so-called country capital".

Mr Cameron, whose remarks were at Downing Street on Monday, said Belgium should always "co-operate with other countries to try to resolve the migrant crisis". He also repeated the call

for external mediators to be involved in figuring out ways to meet the social and economic challenges that are roiling the West.

He told Sky News: ”So I’m not being framed because that sounds like George Murphy. I think what this Chancellor wants to do is to try to carve out a complex structure with other countries to try to resolve things that are less likely than most.”

In his first briefing on refugees a EU spokeswoman said: ”There have been a wave of people attempting to leave Syria or Libya, join Syria, wanted to cross the border to join IS or other Islamic extremist groups or expressed a desire or willing to turn over to join terrorist groups carrying out attacks on civilians.

”Our aim is to broadenen the negotiating principles with all nations to come up with a common structure so that other countries can work together to resolve the crisis across the globe.

”The number of people arriving into Europe has risen more than doubled in the past four years. It is the responsibility of each member to support the handling of the global migrant crisis.

World of freedom

After Monday’s cabinet meeting, Belgium was still grappling to come to terms with a deal that would allow a refugee to cross the EU within two months.

But in his first statement, the chancellor said: ”I assure you that this is the world of freedom: where no matter of ethnicity, race, religion, national origin or national belief we can turn the crisis from one country to another and work together to work together to find a common solution.”

EU refugee and migrant minister Jean-Claud Juncker also said cross-border talks between the 58 members needed to take place before they could come up with a common structure to resolve the crisis.

Belgian immigration minister Step-Luc Lassier warned the long-term effects of the crisis could be ”less likely than most” for refugees fleeing the West and elsewhere across the globe.

More than 5,000 foreigners attempt to cross the EU every day from Syria and Iraq - many of them come from Afghanistan and Pakistan, which accounts for nearly a third of the EU’s total flights, according to the UN Office for Refugees.

UNICE figures show there are 450,000 foreigners attempting to cross the EU from Syria and Iraq every day.

<|endoftext|>

A.5 Qualitative XSum Examples

To give a concrete view of AURORA-LM’s behavior on XSum, we include two examples from the evaluation set. Each example pairs the complete input article with its reference summary and the summary produced by our model. The ROUGE-F1 values shown alongside the examples are computed for the individual pairs; the main quantitative comparison is based on the corpus-level results in Table 1(b).

Example 1 ROUGE-F1 (R-1 / R-2 / R-L): 60.5 / 34.1 / 55.8

Source. After a drab game with few chances for either side, Mohamed Salah teed up El Said, who struck firmly past Uganda keeper Denis Onyango on 89 minutes. Uganda’s Joseph Ochaya earlier had a goal correctly disallowed for offside. The win means Egypt play Ghana on Wednesday with top spot of Group D still at stake, while Uganda are out of their first tournament since 1978. Ghana secured their place in the quarter-finals with a 1–0 victory over Mali earlier on Saturday. On a deteriorating pitch in Port-Gentil, seven-times champions Egypt failed to break down a resolute Cranes defence for long periods. Onyango made an impressive clearing header to deny the onrushing Salah at the end of an otherwise lifeless first half. Uganda began the second half brightly, forcing veteran Egypt goalkeeper Essam El Hadary to punch several set-pieces clear but ultimately failed to register a shot on target as Ochaya turned in Faruku Miya’s

through ball from a clearly offside position on 52 minutes. El Said blasted a shot well over when unmarked and Marwan Mohsen went close for the Pharaohs with a header, but Egypt appeared to be heading for their second consecutive goalless draw. However, an otherwise quiet Salah showed great composure late on to check his shot and play in El Said with a reverse pass, the substitute's close-range shot evading Onyango. Uganda now face Mali in the final round of group games on Wednesday, with Mali needing a win and for Ghana to beat Egypt by two goals or more if they are to qualify. Match ends, Egypt 1, Uganda 0. Second Half ends, Egypt 1, Uganda 0. Offside, Uganda. Murshid Jjuko tries a through ball, but Muhammad Shaban is caught offside. Hand ball by Moses Oloya (Uganda). Kahraba (Egypt) wins a free kick in the attacking half. Foul by Hassan Wasswa (Uganda). Offside, Egypt. Kahraba tries a through ball, but Marwan Mohsen is caught offside. Substitution, Uganda. Moses Oloya replaces Denis Iguma. Mohamed Shafy (Egypt) wins a free kick in the defensive half. Foul by Khalid Aucho (Uganda). Goal! Egypt 1, Uganda 0. Abdallah El Said (Egypt) right footed shot from the centre of the box to the centre of the goal. Assisted by Mohamed Salah. Attempt missed. Amr Warda (Egypt) right footed shot from outside the box is too high. Assisted by Mohamed Shafy. Corner, Uganda. Conceded by Ahmed Fathy. Attempt blocked. Joseph Ochaya (Uganda) left footed shot from outside the box is blocked. Assisted by Nicholas Wadada. Delay over. They are ready to continue. Substitution, Uganda. Nicholas Wadada replaces Faruku Miya. Substitution, Egypt. Kahraba replaces Trezeguet. Delay in match Khalid Aucho (Uganda) because of an injury. Attempt saved. Marwan Mohsen (Egypt) right footed shot from outside the box is saved in the centre of the goal. Assisted by Amr Warda. Tonny Mawejje (Uganda) wins a free kick in the defensive half. Foul by Mohamed Elneny (Egypt). Delay over. They are ready to continue. Delay in match Mohamed Shafy (Egypt) because of an injury. Foul by Muhammad Shaban (Uganda). Ali Gabr (Egypt) wins a free kick in the defensive half. Hand ball by Trezeguet (Egypt). Attempt missed. Marwan Mohsen (Egypt) header from the centre of the box is just a bit too high. Assisted by Mohamed Shafy with a cross following a set piece situation. Foul by Murshid Jjuko (Uganda). Marwan Mohsen (Egypt) wins a free kick in the attacking half. Godfrey Walusimbi (Uganda) wins a free kick on the left wing. Foul by Ahmed Fathy (Egypt). Attempt saved. Trezeguet (Egypt) right footed shot from outside the box is saved in the centre of the goal. Attempt blocked.

Generation. Egypt's Abdallah El Said scored an injury-time winner to beat hosts Uganda and reach the Africa Cup of Nations quarter-finals.

Reference. Substitute Abdallah El Said scored a late winner for Egypt to knock Uganda out of the Africa Cup of Nations.

Example 2 ROUGE-F1 (R-1 / R-2 / R-L): 53.3 / 35.7 / 53.3

Source. The latest picture was taken by the rover at "Buckskin," which is the seventh rock target on its mission. Curiosity does this planetary photography in the same way we would take a selfie - by holding a camera at arm's length and framing itself. The pictures must then be stitched together to make this final scene. Nasa is trying to take one at every location where the robot drills into the surface of the red planet. The robot has now been on Mars for three Earth years. It is in what's known as Gale Crater. Curiosity is currently climbing through what's known as Mount Sharp, examining the rocks as it goes. Scientists want to understand when and how Mars became so barren.

Generation. The US space agency, Nasa, has taken a stunning picture of the Mars robot.

Reference. The US space agency Nasa has issued a "selfie" portrait from its Curiosity rover on Mars.

A.6 Few-Shot Benchmark Demonstrations

To give a concrete view of our 1B-parameter AURORA-LM's few-shot evaluation protocol, we list the two-shot demonstrations used for each benchmark task in Sec. 4.1. The quantitative comparison is reported in Table 2.

Table 17: Two-shot demonstrations used for each benchmark task.

Example 1	Example 2
HellaSwag	
Context: The girl puts the bread into the toaster and pushes the lever down. The bread (A) becomes a slice of pizza. (B) starts to toast and turn brown. (C) disappears immediately. (D) turns into a glass of water. Answer: starts to toast and turn brown.	Context: The goalkeeper sees the ball coming towards the net. He dives and (A) catches the ball with his hands. (B) starts dancing in the field. (C) opens a laptop to check email. (D) runs away from the stadium. Answer: catches the ball with his hands.
MMLU	
Question: Which gas do plants absorb from the air during photosynthesis? (A) Oxygen (B) Carbon dioxide (C) Nitrogen (D) Hydrogen Answer: Carbon dioxide	Question: What is the capital of France? (A) London (B) Berlin (C) Paris (D) Madrid Answer: Paris
OpenBookQA	
Question: Which tool is best for tightening a screw? (A) spoon (B) hammer (C) screwdriver (D) paintbrush Answer: screwdriver	Question: What do plants absorb from the air during photosynthesis? (A) carbon dioxide (B) oxygen (C) helium (D) salt Answer: carbon dioxide

Two-shot demonstrations (continued).

Example 1	Example 2
RACE	
Article: Mary went to the store to buy some fruits. She bought five apples and two oranges. She paid 5 dollars in total. Question: What did Mary buy? Options: (A) Bananas (B) Apples and oranges (C) Grapes (D) Watermelon Answer: Apples and oranges	Article: Sarah loves reading books. She goes to the library every Saturday morning. Last Saturday, she borrowed three mystery novels and two science fiction books. Question: How many books did Sarah borrow last Saturday? Options: (A) Two (B) Three (C) Four (D) Five Answer: Five
SIQA	
Question: Jordan wanted to tell a joke to his friends. What does Jordan need to do before this? (A) ignore his friends (B) think of a funny story (C) leave the room Answer: think of a funny story	Question: Tom forgot his umbrella on a rainy day. How would Tom feel? (A) happy (B) excited (C) frustrated Answer: frustrated

Two-shot demonstrations (continued).

Example 1	Example 2
SQuAD	
Context: The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the 10th and 11th centuries gave their name to Normandy, a region in France. They were descended from Norse raiders and pirates from Denmark, Iceland and Norway. Question: In what country is Normandy located? Answer: France	Context: The Apollo program, also known as Project Apollo, was the third United States human spaceflight program carried out by NASA, which accomplished landing the first humans on the Moon from 1969 to 1972. Question: Which organization carried out the Apollo program? Answer: NASA

Two-shot demonstrations (continued).

Example 1	Example 2
StoryCloze	
Story: I wanted to make an omelet. I cracked two eggs into a bowl and whisked them. Then I poured them into a hot pan. (A) I ate a delicious omelet for breakfast. (B) I decided to order a pizza instead. End: I ate a delicious omelet for breakfast.	Story: The runner tied his shoes tight. He sprinted as fast as he could during the race. He crossed the finish line first. (A) He was sad that he lost the race. (B) He won the gold medal. End: He won the gold medal.
ARC-Challenge	
Question: Which of the following best explains why a helium balloon floats in air? (A) Helium is lighter than the rubber of the balloon. (B) Helium has a lower density than air. (C) Helium atoms are larger than air molecules. (D) Helium reacts with the air to produce lift. Answer: Helium has a lower density than air.	Question: Which process is responsible for the movement of water from plant roots to the leaves? (A) Active transport only (B) Gravity (C) Transpiration pull (D) Osmosis only Answer: Transpiration pull
WinoGrande	
Sentence: The cup didn't fit in the suitcase because _ was too big. (A) the cup (B) the suitcase Answer: the cup	Sentence: The trophy didn't fit into the suitcase because _ was too small. (A) the trophy (B) the suitcase Answer: the suitcase